App Studio report: Ghost
Aron Hammond | 10437215

Structure

My application is comprised of 8 vies with corresponding controllers. These controllers are all wrapped in one NavigationViewController. In addition there are 3 model classes and 2 helper classes.

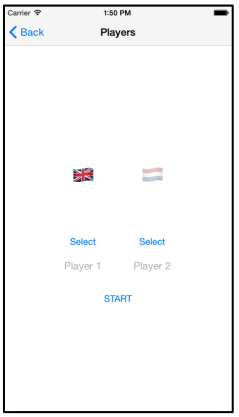
View Controllers:

HomePageViewController:
This is the first screen on start up. There are 3 UIButtons controlling segues to either the NewGameViewController, SettingsViewController or HighscoreViewController. Also, in the ViewDidLoad() method, a global instance of the Game class is initialized.

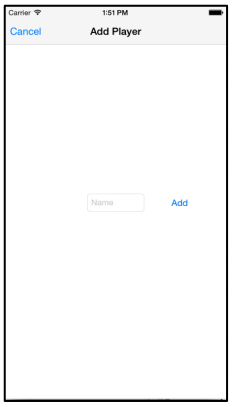
SettingsViewController:
Simple settings screen where the user can toggle the default language of the game by touching a UIButton. If no default is set, it is set to English initially. The UIButton's title flips between the flag's of the current default language

Hurdle: coming up with an efficient way to 'flip' languages based on the current setting. Solution: ternary conditional operation: "english ? setTo(dutch) : setTo(english)

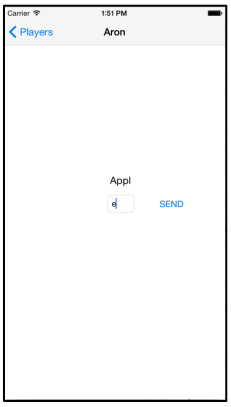
ChoosePlayerViewController:
A UITableViewController class. The table view shows the list of players stored in the Settings.sharedInstance. It also recognizer a left swipe on a cell, which shows a delete button.
If the user wants to add a new player, he or she can do so by tapping the "+" on the navigation bar.



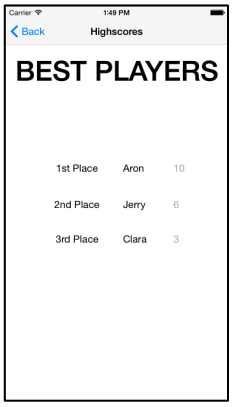
NewGameViewController:
This view lets the user pick 2 players that are going to play the game, and set the language for this specific game. The two UIButtons labeled 'SELECT' both trigger a segue to the ChoosePlayerViewController. Either with identifier "Choose Player 1" or "Choose Player 2". The "START" button triggers a segue to the GamePlayViewController on the condition that both players are selected



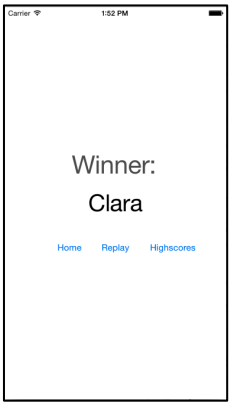
AddPlayerViewController:
Contains a UITextField where the name of the new player can be specified. UIButton "Add" calls a function addPlayer(name: String) in the Settings class with the content of the UITextField as an argument.



GamePlayViewController:
Elements: UILabel, UITextfield and UIButton. The "SEND" button triggers the function playLetter() which sends the contents off the UITextField to the Game class. No input is validated by the controller, this happens in the model. The function also updates the screen and asks the model if the game has ended. If the game ended, a segue is triggered to the FinishedGameViewController.



HighscoreViewController:
Shows the tree players with the most wins. List consists of 9 UILabels, 3 for the place, 3 for the names and 3 for the scores.
The data comes from an object of the HighscoreList class.



FinishedGameViewController:
This view is pushed on the stack of the navigation controller, but the navigation bar is hidden programmatically for esthetic reasons and to make it unable to go back to the GamePlayView. In the viewDidLoad() method, the game model is asked to reset the underlying dictionary.
Also contains 3 UIButtons: “Home” pops to the rootviewcontroller, “Replay” pops back to the GamePlayView and “Highschores” shows the HighscoreView.

Classes:

Player:
Represents a player in the game. A Player’s properties are name (String) and a score (Int). These objects are used in the Game model to represent the current players and by the highscore list to show the top 3 best players.

HighscoreList:
Class that exists to generate a sorted copy of the list of the list of players owned by Settings.

Game:
Model class that holds all of the rules and the functionality of the gameplay. To comply with the MVC paradigm, I had to look for a way to share the model data between multiple view controllers. To solve this problem I choose to make use of a Singleton-pattern. That way it is possible for every controller/class to access the same instance Game, so the data can be used to set UI elements appropriately.
A notable design decision for this class was the choice to save the current players in an array. This way if both players need to be used in the code, it can be done by iterating over this array. This leaves this part of the code independent of the number of players. To facilitate this, the suggested turn() function had to be replaced by what is now currentPlayerIndex(). This function returns the index at which the player who now guessing. As a consequence, winner() was changed to return the player at the index after the player who lost (this wouldn’t make any sense when the number of players is larger than two, so this part is not so independent).

DictionaryClass:
First and foremost: the name of this class contains class because of some errors that occurred because “Dictionary” is a keyword in swift.
Because of the costs of reading/splitting the dictionary text files, on the first run of the app the array representations of the dictionaries are stored in NSUserDefaults. On following launches, they are returned from the defaults and the ‘dict’ property is set to de default language.

Settings:
Holds information about the players and handles those as well. It also manages the default language and in future versions all other editable settings of the application, like sound effects and alternative rules.

Experience

Hurdles:

- The Swift programming language
- Data persistence
- State preservation

A big hurdle for this project was my limited knowledge on the Swift programming language. There were a lot of situations to which I knew what needed to be done but I just didn't know the Swift way. But step by step it sinked in and I also got better at navigating the internet and the official documentation to find implementation details.
Two subjects I struggled with the most are *data persistence* and *state preservation.* Initially I dived deep into archiving to find some solution to for these issues. After some failed attempts to archive the Game instance, I decided to just store game specific properties in NSUserDefaults. State preservation seemed very easy to implement initially, and I got it working in the simulator, but after hours of googling it is still not functioning properly on a physical device.

Gained insights:

- Object oriented programming
- Apple's documentation
- App development process

Also the concept of object oriented programming was new to me. But after reading about it and using it to create this project I feel that it is a very natural way of programming. Also working by the MVC approach that goes along with it very well, albeit being less intuitive.
Learning to make use of the documentation was a challenge because all entries are very elaborate, assume knowledge across the platform and the more advanced the functionality, the less examples are given. But eventually (also because the platform became more familiar) the documentation became more useful